Hashing Techniques That Allow Dynamic File Expansion
Data Structure of the Extendible Hashing:

- Class Directory {
  - int d = 2;  // d is called global_depth initialized with 1, 2 or any integer.
  - // a pointer to array of inter with 2d elements, each holding
  - // the address of a bucket.
  - int * addresses = new int[2d] { bucket-addresses};
  - }

- class Bucket<T> {
  - int d';  // local_depth in the bucket.
  - int rec# = 0;  // number of records in the bucket.
  - Static int capacity = 3;  // capacity of a bucket.
  - T array[capacity];  // location for holding records.
  - }

Directory and Buckets

- Directory holds integer $d$, called global-depth, and a table of bucket addresses. The table size is determined by $2^d$. Each item of the table holds bucket one address.

- The bucket holds a $d'$, the local-depth, capacity and actual number of records held and the actual records.
Global and Local Depth

- The global depth, \( d \), is used to determine the numbers of bits in record’s hash code to find a bucket for the record. The \( d \)-bits of records' hash code can be the most significant or least-significant bits. Least-significant is more convenient to use.

  - Given \( d = 2 \), and a record's hash code = 13 (1101), the 01 or \( (13 \mod 2^2 = 1) \) will be used to find out the bucket address at directory.address[1].

- The local depth, \( d' \), has similar meaning of global depth, and it starts with the same value as current global depth. The global depth will change when more space is needed.
capacity = 3;  // capacity of a bucket

- 4, 6, 7, 10, 16, 22, 24, 31
capacity = 3;  // capacity of a bucket

4, 6, 7, 10, 16, 22, 24, 31
capacity = 3;  // capacity of a bucket

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **00**: 2
- **01**: 2
- **10**: 2
- **11**: 2

- **2**: 4, 16, 24
- **2**: 6, 10, 22
- **2**: 7, 31
capacity = 3;  // capacity of a bucket

9, 20, 26

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 2
- 4, 16, 24
- 2
- 2
- 6, 10, 22
- 2
- 7, 31
capacity = 3;  // capacity of a bucket

9, 20, 26
Initial Directory and Buckets

- $\begin{bmatrix} 5 & 13 & 9 & 14 & 26 & 12 & 16 & 7 & 19 & 11 \end{bmatrix}$

- $d = 2$  
- $d' = 2$

- $00 (0) \rightarrow [\ ]$
- $d' = 2$

- $01 (1) \rightarrow [\ ]$
- $d' = 2$

- $10 (2) \rightarrow [\ ]$
- $d' = 2$

- $11 (3) \rightarrow [\ ]$
The initial director and buckets with records with hash codes 10 record inserted.

- \( d = 2 \)  \( d' = 2 \)

- \( 00 \) (0) \( \rightarrow \) \([\ 12 \ 16 \ ]\)

- \( d' = 2 \)

- \( 01 \) (1) \( \rightarrow \) \([\ 5 \ 13 \ 9 \ ]\)

- \( d' = 2 \)

- \( 10 \) (2) \( \rightarrow \) \([\ 14 \ 26 \ ]\)

- \( d' = 2 \)

- \( 11 \) (3) \( \rightarrow \) \([\ 7 \ 19 \ 11 \ ]\)
Split Bucket and Double Directory (to-be-split bucket local depth $d'$ == global depth $d$)

- Add record with hash code = 29, and bucket pointed by address[1] is full and 4 values need to split some into new bucket.
- Before the splitting, we need double the directory, add new bucket and split data.
\( d = 3 \quad \quad \quad d' = 2 \)

1. 000 (0) \( \rightarrow \) [12, 16]
   \( d' = 3 \)
2. 001 (1) \( \rightarrow \) [5, 13, 9] : 5, 9, 29 are moved to new bucket.
   \( d' = 2 \)
3. 010 (2) \( \rightarrow \) [14, 26]
   \( d' = 2 \)
4. 011 (3) \( \rightarrow \) [7, 19, 11]
   pointing to 1st bucket
5. 100 (4) \( \rightarrow \) [00] pointing to 1st bucket
   \( d' = 3 \)
6. 101 (5) \( \rightarrow \) [5, 13, 29]
7. 110 (6) \( \rightarrow \) [10] pointing to the 3rd bucket.
   pointing to 3rd bucket.
8. 111 (7) \( \rightarrow \) [11] pointing to 4th bucket
Notice that when split the records out to new bucket. The hash code must be used to determine whether a record will stay or move to the new bucket. It is not to divided full bucket into two halves.
Splitting without Doubling Directory

- When insertion of a record in bucket pointed by addresses[3], the bucket will be split into 2 without doubling the directory since d' < d.
- Record removal, Bucket Merging and Directory Halving.
  - Two buckets pointed by $0\{x\}^k$ and $1\{x\}^k$, the contents of the 2 buckets can be merged into 1 bucket, return one bucket, reduce the $d'$ by one, and let two director pointers point 1 buckets.
  - When nearly two of $2^k$ pointers point at $2^{k-1}$ blocks, it is the time to halve the directory and, reduce $d$ by one.
Dynamic Hashing: It is a predecessor algorithm of Extendible Hashing Algorithm.

Linear Hashing:
- [https://www.youtube.com/watch?v=Yw1ts57uL7c](https://www.youtube.com/watch?v=Yw1ts57uL7c) part 1.
- [https://www.youtube.com/watch?v=RNoboXmu3zA](https://www.youtube.com/watch?v=RNoboXmu3zA) part 2